KATHOLIEKE UNIVERSITEIT
LEUVEN

Faculty of Economics and Applied Economics

Heuristic procedures for reactive project scheduling

b

Stijn Van de Vonder, Francisco Ballestin, Erik Demeulemeester and Willy Herroelen

DEPARTMENT OF DECISION SCIENCES AND INFORMATION MANAGEMENT (KBI)

KBI 0605

Heuristic procedures for reactive project scheduling

Stijn Van de Vonder[1], Francisco Ballestín[2], Erik Demeulemeester[1] and Willy Herroelen[1]

[1]Research Center for Operations Management,
Department of Decision Sciences and Information Management, K.U.Leuven,
Naamsestraat 69, B-3000 Leuven (Belgium)
Email: <first name>.<last name>@econ.kuleuven.be

[2]Dpto. de Estadística e Investigación Operativa,
Universidad Pública de Navarra
Campus de Arrosadía, 31006 Pamplona (Spain)
Email: francisco.ballestin@unavarra.es

May 22, 2006

Abstract

This paper describes new heuristic reactive project scheduling procedures that may be used to repair resource-constrained project baseline schedules that suffer from multiple activity duration disruptions during project execution. The objective is to minimize the deviations between the baseline schedule and the schedule that is actually realized.

We discuss computational results obtained with priority-rule based schedule generation schemes, a sampling approach and a weighted-earliness tardiness heuristic on a set of randomly generated project instances.

Keywords: Project scheduling, uncertainty, stability, reactive scheduling

1 Introduction

During project execution, project activities may take longer or shorter than initially expected, resources may become temporarily unavailable, new activities may have to be included, the project may have to be interrupted for a certain time, etc. Effectively dealing with these uncertainties is an important challenge for any project manager. This paper focuses on processing time uncertainties.

In general, there are two approaches for dealing with uncertainty in a scheduling environment (Davenport and Beck (2002), Herroelen and Leus

(2005)): *proactive* and *reactive scheduling*. Proactive scheduling aims at the construction of a protected initial schedule (*baseline* or *predictive schedule*) that anticipates possible future disruptions by exploiting statistical knowledge of uncertainties that have been detected and analyzed in the project planning phase. However, despite the protection included in the initial schedule, disturbances during project execution may cause deviations from the predictive schedule and may even make it infeasible. Reactive scheduling procedures are then needed to repair the schedule such that it reflects the objectives and constraints of the evolved environment while minimizing the negative impact of the disruption. This paper deals with the development of efficient and effective procedures that can be used during project execution to repair the initial project schedule when needed.

A recent research track deals with the *stochastic resource-constrained project scheduling problem* (*stochastic RCPSP*), an extension of the well-known (deterministic) RCPSP that involves the minimization of the expected makespan of a project with stochastic activity durations (problem $m, 1|cpm, \mathbf{d}_j|E(C_{max})$ in the classification of Herroelen et al. (2000)). The stochastic RCPSP aims at making a project schedule *quality robust*, i.e. insensitive to disruptions that affect the obtained scores on the performance metrics used to evaluate the quality of the schedule (typically the project duration or makespan). Most of the research efforts on the stochastic RCPSP rely on so-called *scheduling policies* (Möhring et al. (1984, 1985)). No use is made of a predictive schedule, but the scheduling problem is viewed as a multi-stage decision process where scheduling decisions have to be made at stochastic decision points that correspond to the completion time of activities, exploiting only knowledge about the observed past and a priori knowledge about the processing time distributions. More recently, Stork (2001) has examined the performance of different classes of policies, while Ballestín (2006) developed efficient metaheuristic solution procedures.

However, in a stochastic environment, ensuring the timely completion of a project for a broad range of scenarios, i.e. quality robustness, is often not the only issue. In many cases where certain preparations have been made once the predictive schedule is established (ordering raw materials, acquiring necessary tools or equipment, organizing the workforce, fixing delivery dates for both subcontractors and customers, etc.), it is desirable that the activity starting times that are actually realized during project execution differ little from the planned activity starting times in the predictive schedule. *Stability* or *solution robustness* refers to the insensitivity of planned activity start times to schedule disruptions that may occur during project execution.

The benefits of generating stable predictive schedules have been demonstrated by Van de Vonder et al. (2005a). However, solution robustness or schedule stability should also be maintained when the predictive schedule breaks and needs to be repaired. Reactive procedures should try to repair the predictive schedule in such a way that the safety included in the original predictive schedule is preserved.

The literature concerning robust reactive project scheduling is virtually void. Yu & Qi (2004) describe an ILP model for the multi-mode RCPSP and re-

port on computational results obtained by a hybrid mixed integer programming/constraint propagation approach for minimizing the schedule deviation caused by a single disruption induced by a known increase in the duration of a single activity.

The objective of this paper is to develop effective and efficient robust reactive project scheduling procedures when multiple activity duration disruptions may occur during the execution of a resource-constrained project. Reactive schedule generation schemes are the subject of Section 2. We briefly review the working principles of existing serial, stochastic serial and parallel schedule generation schemes and develop a new robust serial and a robust parallel schedule generation scheme. We demonstrate that robust serial and parallel schedule generation schemes do not suffer from the so-called Graham anomalies (Graham 1966) when the duration of a single activity is disrupted, while these anomalies may occur when more than one activity is disturbed. The reactive scheduling procedures are developed in Section 3. We describe various priority rules to be combined with the schedule generation schemes described in Section 2 and present a sampling procedure that combines the schemes with multiple priority lists. We also describe a heuristic for the weighted earliness-tardiness problem that may be used for reactive scheduling where the due dates of the activities are their planned completion times in the predictive schedule. The set-up of the computational experiment for validating the various reactive scheduling procedures is described in Section 4. The computational results are analyzed in Section 5. Section 6 is reserved for overall conclusions and suggestions for future research.

2 Robust reactive scheduling

While proactive scheduling efforts are made before the actual start of the project, reactive project scheduling is a multi-stage decision process that takes place during project execution. To construct a predictive project schedule S^0, best practice is to employ a priori statistical knowledge about the stochastic project entities. Mostly, the expected or average duration $E(\mathbf{d}_j)$ of activity j is used to decide upon its predictive starting time s_j^0. The actual duration $\mathbf{d}_j$ of project activity j is only known with certainty at its completion. New information thus becomes gradually available during project execution, possibly requiring a schedule revision. At every point t in time, the *projected schedule* S^t predicts how the project scheduler expects the project to unfold given the information available at that time. If an activity was *projected* to have finished at time t, but it has not, the activity should remain active in the projected schedule. In Van de Vonder et al. (2005b), we explicitly assumed that in that case, the exact realized duration is known and the remainder of the disrupted activity could readily be scheduled for its remaining duration $\mathbf{d}_j - E(\mathbf{d}_j)$. This assumption might be unrealistic and will not be made in this paper. The disrupted activity will be continued for only one time period between t and $t+1$ and will be reconsidered iteratively until it finishes.

When the project finishes at stochastic time T, the projected schedule becomes the *realized schedule* S^T, that provides complete information about the actual realizations of the activity durations. The solution robustness of the project is measured by the deviation $\Delta(S^0, S^T)$ of this realized schedule S^T from the predictive schedule S^0 that was generated before the start of the project.

Solution robust reactive scheduling needs to ensure that the decisions made during project execution result in a small deviation $\Delta(S^0, S^T)$. The objective function used in this paper to evaluate this performance metric is to minimize the weighted expected sum of the absolute deviations between the planned and the realized activity start times, i.e. $\Delta(S^0, S^T) = \sum_j w_j E|s_j^T - s_j^0|$, where s_j^0 denotes the planned starting time of activity j in the predictive schedule S^0, s_j^T is a random variable denoting the actual starting time of activity j in the realized schedule S^T, and the weights w_j represent the disruption cost of activity j per time unit, i.e. the non-negative cost per unit time overrun or underrun on the start time of activity j. We assume that the project will be subject to a predefined project due date δ_n and that the baseline starting time of the dummy end activity s_n^0 is set equal to this δ_n. The cost of finishing the project before δ_n is zero, while the unit cost of surpassing the due date equals w_n.

It should be remarked that evaluating the objective function is very cumbersome, the PERT problem being $\#P$-complete (Hagstrom 1988). In this paper, $\Delta(S^0, S^T)$ will be evaluated using simulation. Not only the evaluation of the objective function, but also the scheduling problem itself is hard. For NP-hardness proofs of several cases of the scheduling problem for stability subject to a deadline and a discrete disturbance scenario, we refer to Leus & Herroelen (2005).

2.1 Schedule generation schemes

Reactive scheduling commonly relies on the application of so-called *scheduling policies* or *scheduling strategies* (Möhring et al. (1984, 1985)) under the objective of minimizing the expected makespan. A scheduling policy can be defined as a decision process that defines which set of activities are started at certain decision points t.

The best-known class of scheduling policies is the class of *priority policies* which order all activities according to a priority list λ and at every decision point select the next activities to start based on this priority list.

Often this selection occurs by applying a *parallel schedule generation scheme* (*parallel* SGS). The parallel SGS iterates over time and starts at each decision time t as many unscheduled activities as possible in accordance with the precedence and resource constraints. The priority list dictates the order in which activities are considered.

In deterministic project scheduling, the *serial* SGS is the best-known alternative for the parallel SGS to decide which activities to start at what decision time. In each iteration, the next unscheduled activity in the priority list is selected and assigned the first possible starting time that satisfies the precedence

and resource constraints. To apply the serial SGS in a stochastic multi-stage decision process such as reactive scheduling, at any decision time a complete schedule has to be generated by applying a deterministic serial SGS. The observed past may not be altered and only activities that have an assigned earliest possible starting time that equals the current decision time are actually started. References to the serial SGS later in this paper, will always imply this *reactive serial SGS*, unless stated differently.

The *stochastic serial* SGS (Ballestín 2006) used in an *activity-based priority policy* (Stork 2001) works as the deterministic serial SGS but adds the side constraint that $s_i \leq s_j$ for any activity i that precedes activity j in the priority list. The stochastic serial SGS might very well result in a different scheduling decision than the deterministic serial SGS. Consider for example the partial schedule shown in Figure 1 for a project with a single renewable resource type. The horizontal bands illustrate the flow of the individual resource units throughout the project. Assume that at decision time 2, activity 3 (with expected duration $E(\mathbf{d}_3) = 3$ and per period resource requirement $r_3 = 2$) and activity 4 ($E(\mathbf{d}_4) = 4$ and $r_4 = 1$) are eligible for scheduling. The stochastic serial SGS with priority list $\lambda = \{1, 2, 3, 4\}$ will not start activity 4 at time 2 because activity 3, a predecessor of 4 in the list λ, has not yet been scheduled. The deterministic serial SGS, on the other hand, will first project activity 3 at time 3 and will decide that this decision does not impede the start of activity 4 at decision time 2.

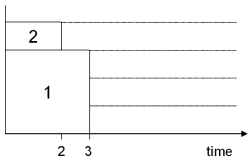

Figure 1: Partial schedule at time 2

Remember that we are primarily interested in robust reactive scheduling procedures, while the previously introduced SGSs are all concerned with minimizing the expected project makespan. Hence, we propose two new SGSs, namely the *robust parallel* SGS and the *robust serial* SGS.

The *robust parallel* SGS operates similarly to the parallel SGS with the side constraint that an activity j is only eligible to be scheduled if the current decision time $t \geq s_j^0$, the starting time of activity j in the predictive schedule. This approach is commonly referred to as *railway scheduling*.

Like the basic serial SGS, the *robust serial* SGS considers activities in the order dictated by a priority list, but instead of starting these activities as early as possible, they will be scheduled at their feasible positions that are the closest possible to their planned starting times in the baseline schedule. The deviation $\varepsilon = \Delta(s_j^t - s_j^0)$ will thus be minimized. Contrary to railway scheduling, this approach allows an activity j to be scheduled earlier than s_j^0. Scheduling activity j at $s_j^0 - \varepsilon$ will even be given priority to scheduling j at $s_j^0 + \varepsilon$ if a tie needs to be

broken between both possibilities with equal ε. The non-retroactivity constraint that will be discussed in Section 2.2.1 dictates that at decision point t, an eligible activity should not be scheduled earlier than t. Whenever $\varepsilon > s_j^0 - t$, the robust serial SGS acts like the serial SGS and searches for the earliest resource and precedence feasible starting time for activity j.

2.2 Properties of schedule generation schemes

In this section we analyze the basic properties of the priority rule-based schedule generation schemes introduced in the previous section. We investigate whether they satisfy the so-called non-anticipativity and non-retroactivity constraints, whether they generate non-delay or active schedules and whether they may suffer from the so-called Graham anomalies (Graham 1966).

2.2.1 Non-anticipativity constraint and non-retroactivity constraint

The *non-anticipativity constraint* (Fernandez & Armacost 1995) specifies that reactive scheduling decisions can only be made on the basis of the observed information from the past and the a priori statistical knowledge of the future. This means that we do not know how long the eligible activities will actually take when we have to decide to schedule them.

As information becomes gradually available during project execution, a scheduling decision made at time t may no longer be the best decision at time $t' > t$. For such cases, a constraint that is complementary to the non-anticipativity constraint will be introduced, i.e. the *non-retroactivity constraint*. This specifies that a reactive procedure may not overrule previous scheduling decisions by scheduling activities in the past. Violations of this constraint may result in infeasible realized schedules.

Assume that we have a project consisting of three activities and a single renewable resource type with constant availability $a = 4$. The activities have expected durations $E(\mathbf{d}_1) = 2$, $E(\mathbf{d}_2) = 2$, and $E(\mathbf{d}_3) = 3$ and resource requirements $r_1 = 2$, $r_2 = 3$, and $r_3 = 2$. Suppose that at time 0 we decided to schedule them by applying a deterministic serial SGS on priority list $\{1, 2, 3\}$ as shown in Figure 2. When at time 2, new information becomes available that reveals that activity 1 will take one extra time unit to complete, the schedule of Figure 3 would minimize the makespan, but violates the non-retroactivity constraint because activity 3 can not be scheduled in the past.

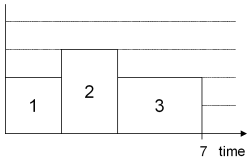

Figure 2: Predictive schedule S^0

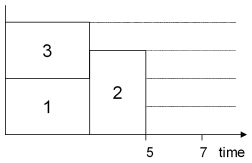

Figure 3: Impossible schedule at time 2

Clearly, all SGSs described in the previous section respect both the non-anticipativity and the non-retroactivity constraint.

2.2.2 Non-delay and active schedules

A feasible schedule is called *active* if for all activities no local or global left shift can be performed. A feasible schedule is called a *non-delay schedule* if for all activities no local or global left shift can be performed even if activities can be preempted at integer time points. A local left shift of an activity j in a schedule is a left shift that can be obtained by successive one period left shifts of the activity, i.e. all intermediate schedules in which the starting time of activity j is successively decreased by one time unit have to be feasible. A left shift which is not a local left shift is called a global left shift (Brucker & Knust 2006).

Kolisch (1996) has shown that for the RCPSP with constant resource capacities and regular measures of performance, any schedule generated by the parallel SGS belongs to the set of non-delay schedules and this set possibly does not contain any optimal solution. On the other hand, any schedule generated by the serial SGS belongs to the larger class of active schedules and this class always includes at least one optimal solution for regular performance measures.

The maximum stability objective function used in this paper, however, is a non-regular performance measure. In our problem setting, an optimal solution minimizes the expected difference between the predictive schedule and the realized schedule ($\sum_j w_j E|s_j^T - s_j^0|$). There is no guarantee that for non-regular performance measures, there always exists an optimal non-delay schedule or an optimal active schedule.

Both the robust parallel and robust serial SGS do not necessarily generate non-delay or active schedules. As with their non-robust variants, the robust serial SGS generates a larger class of schedules than the robust parallel SGS. Still, there does not always exist a priority list λ that would result in the a posteriori optimal realized schedule even if a static scheduling problem with full information about the realized activity duration is solved. For the robust parallel SGS, an activity will never be started before its baseline starting time, while this might well be the case in the optimal solution. For the robust serial SGS, an activity is scheduled as close as possible to its baseline starting time. An activity is only scheduled before or after its baseline starting time *if* scheduling it at its baseline starting time is infeasible.

In short, both robust reactive scheduling schemes try to aggressively repair

the predictive schedule and do not intentionally provide any safety cushion against future disruptions. Protecting a schedule against future disruptions is left as the sole responsibility of the proactive scheduling routine.

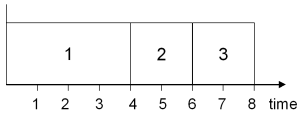

Figure 4: Predictive schedule S^0

Given their scheduling characteristics, the robust parallel and robust serial SGS do not necessarily generate optimal reactive schedules. Consider a small project consisting of three activities that can only be scheduled in series as illustrated in the predictive schedule of Figure 4. If activity 1 finishes early at time 2, both the robust serial and robust parallel SGS would decide to project activities 2 and 3 at their baseline starting times. Assume that $P(\mathbf{d}_2 = 1) = 0.5$ and $P(\mathbf{d}_2 = 3) = 0.5$. This means that there is a 50% probability that starting activity 2 at its predicted start time $s_2^2 = 4$ induces a one period delay in the start of activity 3 with a cost equal to w_3. The total expected cost would then be $0.5(0) + 0.5(w_3)$. However, if we did apply a robust SGS and decided to start activity 2 at time 3, one period earlier than planned, there would have been a 100% probability that we induce a cost w_2, while activity 3 could start as originally planned. If $w_3 > 2 \times w_2$, such a proactive strategy would result in a lower expected stability cost than the cost obtained by either the robust serial or the robust parallel SGS.

2.2.3 Graham anomalies

Given the type of activity duration uncertainty dealt with in this paper, the parallel and serial SGS may suffer from the so-called Graham anomalies (Graham 1966) that were identified for parallel machine scheduling problems under the minimum makespan objective. For example, Figures 2 and 3 illustrate the anomaly that an activity duration extension may result in a shorter makespan if the serial SGS is used in the deterministic RCPSP setting with priority list $\lambda = \{1, 2, 3\}$. A one-period duration extension of activity 1 leads to a two-period reduction in makespan.

In this paper, we are concerned about anomalies that may be induced by activity duration reductions or extensions under the schedule stability objective. Stated otherwise, we are interested to know whether it is possible that an activity duration reduction deteriorates stability or that an activity duration extension improves stability. In order to answer both questions, two cases must be distinguished: (a) the activity disruption is a stand-alone disruption, and (b) other activities are also disrupted (disruption scenario).

Stand-alone activity disruption As long as there is no activity disruption, the projected schedule S^t is, by definition, identical to the predictive schedule

S^0 with a stability cost of $\Delta(S^0, S^0) = 0$. Because the stability cost is always non-negative, no activity duration extension can ever decrease the stability cost, independently of the SGS used. This limits our search for stability anomalies to an investigation of the impact of activity duration reductions on stability.

If we apply non-robust SGSs, almost any duration reduction may generate the stability anomaly because a cost is also incurred if an activity starts earlier than its predictive starting time and its successors may start earlier.

A robust parallel SGS applies railway scheduling. As a result, a stand-alone activity duration reduction does not affect the time that an activity becomes eligible which equals its starting time in the predictive schedule. Because the predictive schedule was both precedence and resource feasible, each activity will be scheduled at its baseline starting time, resulting in a schedule with zero stability cost.

The robust serial SGS decides to plan each activity i at its feasible starting time s_i^t with minimal deviation $|s_i^t - s_i^0|$ from its baseline starting time s_i^0. Because a stand-alone duration reduction never prevents an activity i to start at s_i^0, $|s_i^t - s_i^0| = 0$ for each activity and the stability cost will thus again remain zero.

Clearly, in the single activity disruption case, both the robust serial and the robust parallel SGS do not suffer from stability anomalies induced by changes in activity duration.

Disruption scenario It can readily be shown that an activity duration extension (reduction) can improve (deteriorate) stability if other disruptions already occurred.

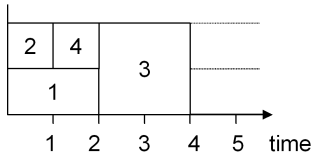

Figure 5: Predictive schedule S^0

Let us illustrate this anomaly on a 4-activity project with a single renewable resource type with availability $a = 2$ and resource requirements $r_1 = 1$, $r_2 = 1$, $r_3 = 2$, $r_4 = 1$. Assume there are no precedence constraints and the expected activity durations are $E(\mathbf{d}_1) = 2$, $E(\mathbf{d}_2) = 1$, $E(\mathbf{d}_3) = 2$, and $E(\mathbf{d}_4) = 1$. Figure 5 shows a baseline schedule with minimum makespan.

Assume that the reactive scheduling schemes use the priority list $\lambda = \{1, 2, 3, 4\}$. If at time 1, activity 2 has not yet finished, the robust SGSs would both result in the projected schedule S^1 of Figure 6, in which only activity 4 does not start at its baseline starting time. The stability cost of this schedule is thus $3 \times w_4$. An extension of activity 1, would result in the projected schedule S^2 of Figure 7 at time 2 with a stability cost of $w_3 + w_4$. It is not difficult to

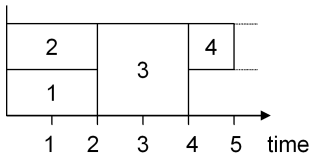

Figure 6: Projected schedule S^1 at time 1

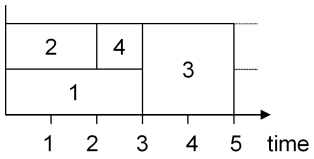

Figure 7: Projected schedule S^2 at time 2

find values for the weights such that this cost will be smaller than the cost of S^1 so that the second activity duration extension would improve stability.

2.2.4 Dispatching

The parallel and robust parallel SGS operate as a *dispatching* or *on-line scheduling* rule, making scheduling decisions dynamically over time. At each decision point, a dispatching rule decides which activities to start without having to decide when the not yet started activities will be projected.

The serial SGS is not commonly used in stochastic scheduling because it does not behave as a dispatching rule. It requires at each decision point t the calculation of a *complete* projected schedule S^t, including best guesses for projected starting times of all the activities that have not yet been started. Full rescheduling at each decision point obviously entails increased computational time. The same reasoning holds for the robust serial SGS. The stochastic serial SGS is based on the serial SGS, but has the advantage that it can be used as a dispatching rule because of its extra constraint that $s_i \leq s_j$ for any activity i that precedes activity j in the priority list.

For a solution robust reactive procedure that tries to minimize the deviation between the projected schedule S^t constructed at current time t and the predictive schedule S^0, having knowledge of the complete projected schedule is essential. Later in this paper, the parallel SGSs will be extended so that they allow for a complete projected schedule to be generated at each decision point.

3 Reactive scheduling procedures

In this section we will describe the four reactive procedures used in the computational experiment of Section 4.

3.1 Priority lists scheduling

The schedule generation schemes described in Section 2.1 were based on a precedence feasible[1] priority list to decide which activities to schedule at each decision time. The following *static priority rules* for generating the priority list will be tested in the computational experiment described in Section 4:

- EBST = earliest baseline activity starting time
- LST = latest starting time
- LW = largest activity weight
- LAN = lowest activity number
- RND = random

The EBST rule orders the activities in non-decreasing order of their starting times in the predictive schedule S^0. The LST priority rule (Alvarez-Valdez & Tamarit 1989), that orders the activities in non-decreasing order of their latest starting time, is included because it ranks among the best priority rules for the deterministic RCPSP. Ties can be broken by ordering the activities in decreasing order of their weights w_j (EBST1 and LST1) or increasing order of their activity numbers (EBST2). The LW rule gives priority to activities with a large disruption cost w_j (smallest activity number as tie-breaker). The LAN rule orders the activities in increasing order of their activity number. The RND rule generates the priority list fully randomly.

Dynamic priority lists depend on the current projected schedule and should thus be updated at each decision time t. The information about the past is known and might influence decisions about the future. We consider two dynamic priority lists:

- EPST = earliest projected starting times
- MC = minimal cost

The EPST rule orders activities at time t by increasing starting times s_i^{t-1} of the projected schedule S^{t-1} generated at the previous decision time. The MC rule orders the activities by increasing $w_i(s_i^0 - t)$. This value will be negative when $s_i^0 < t$ and positive when $s_i^0 \geq t$. For activities with $s_i^0 < t$, priority is given to activities that induce a high stability cost $w_i|s_i^0 - t|$ if started at time t, because delaying their start to a later starting time would even be worse. On the other hand, among the activities with $s_i^0 \geq t$, we prefer to schedule activities with low stability cost first.

[1] A priority list λ is precedence feasible if $\forall(i,j) \in A$: i precedes j in λ

3.2 Fixed resource allocation

As has been proposed by Leus (2003), it is possible to react on schedule disruptions by simply maintaining the resource allocation decisions made on the predictive schedule. For the resource and precedence feasible baseline schedule, a resource flow network can be generated that decides how the resource units are passed on among the activities. A number of resource allocation procedures have been derived in the literature (Leus & Herroelen (2004), Policella et al. (2004) and ongoing research by Deblaere et al. In this paper, we rely on the simple procedure developed by Artigues et al. (2003) that generates a feasible resource flow by extending a parallel schedule generation scheme. Preserving these resource flows when disruptions occur, boils down to right-shifting the affected activities such that the newly generated projected schedule remains feasible.

3.3 Sampling approach

This section describes two reactive scheduling sampling schemes that rely on different priority lists in combination with different schedule generation schemes: basic sampling and time-window sampling. Sampling means in this context that at any decision time several feasible solutions are generated and evaluated and that the best candidate solution is selected.

3.3.1 Basic sampling

The basic sampling approach tries to make a suitable scheduling decision at any decision time t as follows:

for $t = 0, ..., T$ **do**
- Step 1: Check for new scheduling information.
- Step 2: **If** no new information **then** $S^t = S^{t-1}$ and goto period $t+1$ **else** goto step 3
- Step 3: **For** list $l : 1...L$**do**
  - Construct $S^t_{\lambda_l,RP}$ and calculate $\Delta(S^0, S^t_{\lambda_l,RP})$
  - Construct $S^t_{\lambda_l,RS}$ and calculate $\Delta(S^0, S^t_{\lambda_l,RS})$
  - Construct $S^t_{\lambda_l,P}$ and calculate $\Delta(S^0, S^t_{\lambda_l,P})$
  - Construct $S^t_{\lambda_l,S}$ and calculate $\Delta(S^0, S^t_{\lambda_l,S})$
  - Store the projected schedule S^t that minimizes $\Delta(S^0, S^t)$
- Step 4: Start all activities i with $s^t_i = t$.

Step 1 checks for new information becoming available at time t. If at time t, no activity finishes and no activity was projected to finish, then no new information is available compared to the previous decision point $t-1$. The previous projected schedule S^{t-1} remains valid (Step 2).

Instead of using one priority list in combination with one SGS, Step 3 uses multiple lists $\lambda_l \in \{\lambda_1, ..., \lambda_L\}$ at time t in combination with several SGSs. For each of these lists λ_l, a complete projected schedule is constructed using the

robust parallel SGS ($S^t_{\lambda_l,RP}$), the robust serial SGS ($S^t_{\lambda_l,RS}$), the parallel SGS ($S^t_{\lambda_l,P}$) and the serial SGS ($S^t_{\lambda_l,S}$). Doing so, a total of $4 \times L$ candidate projected schedules are generated, with L identifying the number of lists. Among them, the projected schedule S^t that accounts for the smallest deviation $\Delta(S^0, S^t)$ from the predictive schedule is stored. The procedure continues in Step 4 by starting the activities i that have projected starting times $s^t_i = t$ in S^t. Remark that in order to compare the stability costs of all candidate solutions, a complete projected schedule must always be made at any decision time by using the statistical knowledge of future activities. This means that the dispatching advantage of the parallel SGSs can not be exploited.

3.3.2 Time-window sampling

The main problem that might occur in the standard sampling approach is that the decision whether a candidate projected schedule is selected or not might depend on activities that are projected much later in the project at time t'. However, these projected activities are still subject to major uncertainties (*non-anticipativity constraint*) and should not predominate the current decision process. There is no reason to assume that the current reactive policy will also be applied at time t'.

Time-window sampling (*TW Sampling*) tries to cope with this problem by making use of a time window (TW). The difference with the basic sampling approach lies in the generation of the candidate projected schedules $S^t_{\lambda,.}$ at time t (Step 3). Instead of generating a complete projected schedule by applying the current SGS on the current list λ, *TW Sampling* only applies this policy to decide which activities to project within a certain time window $[t, t + \Theta]$. Activities that are not planned to start within this time window, are projected by following the priority list $\lambda_1 = EBST1$. The generation scheme to transform this priority list into a projected schedule is the robust variant of the SGS applied within the time window. Note that deciding which activities to project within $[t, t+\Theta]$ already requires a complete schedule if we apply a non-dispatching SGS such as the serial SGS. For on-line scheduling procedures, such as the parallel SGS, a complete projected schedule is only required once. The dispatching advantage of the parallel SGSs that was absent in the basic sampling procedure can be exploited in *TW Sampling*.

The results of *basic* and *TW sampling* depend on the number of priority lists L used and the actually selected priority lists λ_l. For *TW Sampling* the time window size Θ will be an important parameter. The impact of these settings will be examined in Section 5.

3.4 A heuristic WET procedure

The reactive scheduling problem at each decision point can be viewed as a *Resource-Constrained Project Scheduling Problem with Weighted Earliness-Tardiness Costs* (problem $m, 1|cpm|early/tardy$ in the notation of Herroelen et al. (2000)). Due dates are set equal to the activity completion times $s^0_j + E(\mathbf{d}_j)$

in the predictive schedule. The earliness and tardiness costs will be symmetrical and used as the weights w_j in the stability objective function, except for the earliness cost of the dummy end activity which will be set equal to zero.

Some efficient exact procedures for solving problem $m,1|cpm|early/tardy$ have been proposed in the scheduling literature (Schwindt (2000), Vanhoucke et al. (2001), Kéri & Kis (2005)). However, Van de Vonder et al. (2005b) showed that calling an exact weighted earliness-tardiness procedure at any schedule breakage point is already computationally infeasible for small network sizes.

Recently, Ballestín & Trautmann (2006) developed a population-based iterated local search algorithm for the weighted earliness-tardiness resource-constrained project scheduling problem with minimum and maximum time lags (problem $m,1|gpr|early/tardy$). Although faster than exact procedures, this procedure still remains computationally demanding for our problem. We now introduce an adapted version of this procedure which is based on the metaheuristic *Iterated Local Search* (Lourenço et al. 2002). It is a version of the algorithm described in Ballestín & Trautmann (2006), customized to the special characteristics of the problem. Some of the original features have been omitted to reduce the computational requirements of the algorithm.

The algorithm runs as follows:

WET_Procedure

1. Initialize Elite Set.
2. **While** without_imp < max_without_imp **do**
   a. Select a schedule S from Elite Set.
   b. S' = Perturbation2(S).
   c. S" = Local_Search(S').
   d. **If** $\Delta(S'', S^0) < \Delta(S^+, S^0)$ **then** {without_imp = 0, $S^+ = S''$}.
   e. **Else** without_imp = without_imp+1.
   f. **If** $\Delta(S'', S^0) < \Delta(S^-, S^0)$ **then** S" replaces S^- in Elite Set.
3. Return the best solution obtained: $S^t = S^+$.

S^+ and S^- are the best and worst solution of Elite Set respectively.

Step 2 is repeated iteratively until the number of iterations without improvement (without_imp) equals a predefined number (max_without_imp). Define *card* as the cardinality of the Elite Set. At any time, the *card* best solutions are stored. We initialize the Elite Set in Step 1 as follows:

Initialize Elite Set(card, nitial, nitial2)

1. $S = S^t_{\lambda_{EBST1},RS}$. $S2 = S^t_{\lambda_{EPST},RS}$.
2. S' = Local_Searches(S). S2' = Local_Searches($S2$).
3. Temporary Set = {S',$S2$'}. Elite Set ={ }.
4. **For** $i = 0, i < nitial2 - 2$ **do**
   a. **If** i is even **then** λ = Perturbation1(λ_{EBST1}).
   b. **Else** λ = Perturbation1(λ_{EPST}).
   c. Construct $S_i = S^t_{\lambda,RS}$ and calculate $\Delta(S^0, S_i)$.
5. Temporary Set = Best *nitial* solutions from the *nitial*2 initial solutions.

For $i = 0, i < nitial$ **do**
   a. Restore S_i from Temporary Set

b. S' = Local_Search(S_i).
c. Elite Set = Elite Set $\cup$ {S'}.

6. Elite Set = Best *card* solutions from the *nitial* initially generated solutions.

In Perturbation1, *nunsche*/10 activities are chosen one after the other and reintroduced in the activity list elsewhere, where *nunsche* is the number of unscheduled activities. The movements are made so that the outcome is an activity list. Perturbation2 delays the position of some advanced activities and advances the position of some delayed activities in the activity list. It is described in more detail under the name of Perturbation4 in Ballestín & Trautmann (2006). Local_Search includes three of the Local Searches used in that paper. The second one sorts the early activities in decreasing order of their finish times in the solution S and schedules each activity as closely as possible to its due date. Then it proceeds analogously with the delayed activities, sorting them in increasing order of their starting times in S. Before applying this local search, we employ LocalSearch 1, which allows more freedom in the movement of activities. Each early (delayed) activity is scheduled as late (early) as possible, but the movement is stopped right before the objective function deteriorates. For each delayed activity that can be left shifted due to precedence relationships, the last local search calculates the set of activities B that restrain this movement and that can be moved. The function considers each activity $j \in B$ and unschedules it. Then it calculates whether it is better to schedule first i and then j, each of them as close as possible to its real due date. After working with all the activities of B, the method performs the best of these movements if it produces an improvement in the objective function.

4 Experimental set-up

All algorithms have been coded in Microsoft Visual C++ 6.0 and have been tested on the 600 120-activity instances of the well-known PSPLIB 120 data set (Kolisch & Sprecher 1997).

Two types of predictive schedules S^0 are used as input for the reactive procedures. The first type is generated using the combined crossover algorithm for the RCPSP developed by Debels & Vanhoucke (Debels & Vanhoucke 2006) [2] with 50,000 schedule generations as stop condition. This algorithm has been shown to be among the best performing metaheuristic RCPSP procedures for both small and larger data sets. The obtained schedule has a makespan of $C_{\max}$ and can be regarded as quality robust for a stochastic environment. The second type of predictive schedule is derived from the first by applying the proactive STC heuristic (Van de Vonder et al. 2005a) for inserting time buffers. In both predictive schedules, the project due date δ_n will be set equal to $\lfloor 1.3 \times C_{\max} \rfloor$, which was found to be adequate for most project schedules in a recent study by Van de Vonder et al. (2006).

[2] An executable program can be found at www.projectmanagement.ugent.be/downloads/RCPSP

Realized activity durations $\mathbf{d}_j$ are drawn from a beta-distribution with parameters 2 and 5, with mean equal to the expected activity duration $E(\mathbf{d}_j)$ and with a variance depending on the degree of uncertainty included in $\mathbf{d}_j$. We randomly determine for every activity whether it has *high*, *low* or *medium duration variability*. For more details on the generated test instances, we refer to Van de Vonder et al. (2005a).

The stability cost $\sum_{j \in N} w_j E|s_j^0 - s_j^T|$ is evaluated by drawing the w_j for each activity $j \in \{1, 2 \ldots n-1\}$ from a discrete triangular distribution with $P(w_j = q) = (21 - 2q)\%$ for $q \in \{1, 2....10\}$. This distribution results in a higher probability for low weights and in an average weight $w_{avg} = 3.85$. The weight w_n of the dummy end activity denotes the marginal cost of violating the project due date and will be fixed at $\lfloor 10 \times w_{avg} \rfloor = 38$. For an extensive evaluation of the impact of the activity weight of the dummy end activity, we refer to Van de Vonder et al. (2005b) and Van de Vonder et al. (2006).

Extensive simulation has been used to evaluate all procedures on stability and computational efficiency. For every network instance the average stability cost over 100 execution runs has been computed.

5 Computational results

All computational results have been obtained on a Pentium IV 2.4 GHz personal computer. In this section, results of the reactive procedures will be discussed for the two predictive schedules described in the previous section. Section 5.1 discusses results when the metaheuristic RCPSP solution is used as a quality robust predictive schedule, while Section 5.2 examines the impact of the solution robust predictive STC-schedule on stability.

5.1 Quality robust predictive schedule

Table 1: Average stability cost on PSPLIB 120

λ	RP	RS	P	S	Avg.
EBST1	2457.69	2481.89	3782.89	2520.68	2810.79
EBST2	2493.39	2516.89	3810.77	2543.05	2841.02
LAN	2889.71	5302.13	5833.49	6903.29	5232.15
LST1	3578.85	6691.31	7858.82	9557.33	6921.58
LW	3332.21	7338.43	7707.27	10312.68	7172.65
RND	3345.38	7256.90	7553.44	9985.12	7035.21
EPST	2560.37	2910.73	3862.35	3064.84	3099.57
MC	2959.78	2856.34	6457.86	3128.49	3850.62
Avg.	2952,17	4669,33	5858,36	6001,93	

In this section, the predictive schedule is generated by the combined crossover algorithm of Debels & Vanhoucke (2006). Table 1 shows us the average stability costs over 100 iterations for the 600 instances of the PSPLIB 120 set. We remark that the priority list should be ordered by increasing starting times (EBST1, EBST2 or EPST) to obtain good results. EBST1 slightly outperforms EBST2 as the best priority list. Despite the extra computational time spent by dynamical lists to constantly update the priority list based on the new information acquired in the current projected schedule, they do not obtain better results. After all, we try to minimize the deviation between S^T and the predictive schedule S^0, not the projected schedules. Applying the robust parallel SGS on the list λ_{EBST1} obtains the best overall results. The robust serial SGS obtains very similar results on λ_{EBST1}, but the results of this SGS are more sensitive to the priority list that is applied. For example, λ_{LW} increases the average stability cost with 36% compared to λ_{EBST1} when we apply the robust parallel SGS, while this increase is as high as 196% for the robust serial SGS.

Table 2: Computational times on PSPLIB 120 (in s.)

	RP	RS	P	S
Static	0.04	0.52	0.04	0.72
Dynamic	3.37	0.96	3.76	1.18

Table 2 shows us the computational requirements of the priority list policies. Static priority lists combined with a parallel SGS are computationally the most efficient procedures among the priority policies. They take on average only 0.04s per PSPLIB 120 instance (for 100 execution scenarios). Due to the absence of the dispatching property discussed in 2.2.4, serial SGSs consume substantially more time, i.e. 0.62s on average per network (average of 0.52 and 0.72 in Table 2). Dynamic priority lists require extra computational time to recalculate the priority list at every decision time. Applying a serial SGS on a dynamic list almost doubles the computational time needed to 1.07s. For the parallel SGS the impact of dynamic lists on computational efficiency is even far more pronounced because the dispatching property can no longer be exploited and the parallel SGS needs then to search the priority list at every decision time, while the serial SGSs only do this once. The parallel SGSs require on average 3.56s to solve the PSPLIB instances. Robust SGS are on average slightly faster than their non-robust counterparts. We obtain the very uncommon, but promising conclusion that the best priority policy, applying the robust parallel SGS on λ_{EBST1}, is also among the fastest.

The results and computational requirements for fixed resource allocation and the stochastic serial scheduling scheme are given in Table 4. Fixing the resource flows by the algorithm of Artigues et al. (2003) scores substantially worse on stability than the better results of priority list scheduling. On the PSPLIB 120 data set, we obtain an average stability cost of 4869.88 in less than

Table 3: Benchmark results

	$\sum_{j \in N} w_j E\lvert s_j^0 - s_j^T \rvert$	Time (s)
Fixed resource allocation	4869.88	0.0060
Stochastic serial SGS on λ_{EBST1}	4222.77	0.0097

0.01s. Delays are propagated throughout the project because of the pre-decided resource flows. Even imposing the railway scheduling property does not improve stability cost because it entails even more delay. Applying the stochastic serial generation scheme on λ_{EBST1} also delivers unsatisfying results with an average stability cost of 4222.77.

Table 4: Advanced heuristics

	$\sum_{j \in N} w_j E\lvert s_j^0 - s_j^T \rvert$	Time (s)
Sampling	2217.97	23.0
TW Sampling	2172.74	32.5
ILS	2073.53	79.5

Next we will investigate the results obtained by the more advanced heuristics described in this paper, being *Sampling*, *TWsampling* and ILS. An overview can be found in Table 4.

The performance of the sampling procedure highly depends on the lists λ_l that we include. We experimentally decided to include the six static priority lists of Table 1 and their precedence feasible backward lists (BW). A backward list is a precedence feasible priority list with reverse priorities for the activities. The backward list of the $EBST$ rule is thus the Latest Baseline Starting Time rule, etc. These backward lists might seem illogical as robust reactive procedures but will prove their use in the sampling procedure. The logic behind their inclusion is that when a priority list does not result in a schedule with low deviation from S^0, its inverse list might do so. The backward list of λ_{EBST2} has been removed from consideration because it hardly improved any results due to its resemblance with $EBST1$ BW.

The number of included list L thus equals 11, resulting in $4 \times L = 44$ candidate projected schedules at any decision time. The average stability cost on PSPLIB 120 is 2217.97, which is a 9.8% decrease compared to the best priority list policy of Table 1. The sampling procedure runs on average in 23s. The average makespan is 161.4. At on average 76.9 decision times during execution, new information is available and rescheduling required. Table 5 shows the frequencies that any policy generates a candidate projected schedule with minimum stability cost. In 52 % of all decision times, (λ_{EBST1}, RS) generates a candidate projected schedule that is not dominated by any of the other policies.

In most cases several policies will result in projected schedules that are either identical or have equal stability cost. The ordering of the policies is important because a policy will only be selected if its projected schedule has a lower stability cost than all previously considered schedules. We start by considering all projected schedules generated by the robust parallel SGS, ordered by increasing numbers as indicated in Table 5. Afterwards the robust serial, parallel and serial SGS will be considered respectively. Table 6 shows the percentages that each policy is actually selected as best for rescheduling by the sampling procedure.

Table 5: Frequencies that policies generate best candidate

λ	RP	RS	P	S
1. EBST1	37.05%	52.01%	18.88%	35.03%
2. EBST2	33.05%	46.22%	18.14%	31.95%
3. LAN	25.70%	20.58%	15.88%	15.24%
4. LST	22.87%	19.58%	15.28%	14.64%
5. LW	24.17%	19.92%	15.61%	14.86%
6. RND	23.33%	19.48%	15.31%	14.63%
7. EBST1 BW	23.01%	19.45%	15.29%	14.61%
8. LAN BW	23.34%	19.54%	15.42%	14.67%
9. LW BW	22.91%	19.42%	15.28%	14.61%
10. RND BW	23.98%	19.88%	15.53%	14.87%
11. LST BW	23.26%	19.45%	15.37%	14.58%

The best simple priority policy (λ_{EBST1}, RP) is considered first and selected in 37.05% of all cases. In 62.95% of all decisions, at least one other policy yielded better results. Aggregated over all SGSs, the EBST1 priority rule was best in 64.03% of all decisions made. More importantly, however, is that also the overall less performing policies, such as the non-robust ones and the ones that follow backward priority lists were selected. In 6.62% of the procedure calls, a non-robust SGS is preferred. The seemingly illogical backward lists outperform all the logical forward lists in almost 9% of the cases. Excluding them would deteriorate results substantially.

TWsampling yields better results than basic sampling, but requires an extra parameter setting. There is no single time window size that performs well for all project instances. $\Theta = 5$ can on average be considered as a good time window size. It results in an average stability cost of 2172.74, which is a 2% decrease in stability cost compared to basic sampling. Because the serial SGSs require the construction of two complete schedules at any decision time, i.e. one to decide which activities to start in $[t, t + \Theta]$ and one to complete the projected schedule, *TWsampling* requires more computational time (32.5s) than basic sampling. The selection frequencies of *TWSampling* differ slightly from these of *Sampling* that were shown in Table 6. Most remarkable is an increase in selections of $EBST1$ (71.80%) to the detriment of $EBST2$ (4.39%). $EBST2$ is only selected if it is better and thus by definition different from $EBST1$.

Table 6: Selection frequency of all policies by *Sampling*

λ	RP	RS	P	S	Total
1. EBST1	37.05%	23.16%	1.11%	2.72%	64.03%
2. EBST2	4.94%	5.81%	0.40%	1.33%	12.48%
3. LAN	4.99%	1.02%	0.21%	0.06%	6.28%
4. LST	2.06%	0.45%	0.09%	0.01%	2.61%
5. LW	2.79%	0.54%	0.13%	0.03%	3.49%
6. RND	1.99%	0.32%	0.09%	0.01%	2.42%
7. EBST1 BW	1.56%	0.33%	0.09%	0.02%	1.99%
8. LAN BW	1.35%	0.23%	0.08%	0.02%	1.68%
9. LW BW	1.35%	0.25%	0.07%	0.01%	1.67%
10. RND BW	2.05%	0.37%	0.09%	0.02%	2.52%
11. LST BW	0.65%	0.12%	0.04%	0.01%	0.82%
Total	60.78%	32.61%	2.39%	4.23%	

In *TWSampling*, both rules will often result in the same projected schedule. Backward lists are selected in 7.18% of the decision times, which is a substantial decrease that is equally spread over all backward lists.

The trade-off between performance and computational requirement of the Iterated Local Search (ILS) procedure also depends highly on the parameter settings. We experimentally set $card = 3$, $nitial = 10$ and $nitial2 = 50$. The obtained average stability cost is 2073.53 in 79.5s per network. This is a 15.6% improvement compared to the best priority list policy and a 6.5% improvement compared to *Sampling*. Rescheduling is invoked on average 76.9 times. Remark that traditional WET procedures would be computationally infeasible. A WET procedure that solves the RCPSP-WET in 1s on average on a 120-activities network, will without any doubt be considered as computationally efficient. However, in our simulation experiment this procedure has to be called on average 76.9 times for 100 execution scenarios, which would take us more than two hours per network! For project managers that rely on simulation methods to evaluate the predictive schedule of their project, computationally demanding metaheuristic or exact WET procedures are obviously not suitable as reactive scheduling policies.

ILS generally needs some time to substantially improve results compared to the priority list policies. At first, all priority lists in the Elite Set will be situated in the neighborhoods of $EBST1$ and $EPST1$. *Sampling* directly compares totally different solutions and will improve stability even if the number of lists L is very small. On the other hand, local search algorithms continue improving if more time is allocated, while adding more lists to sampling does not necessarily result in an improvement. It is challenging to add priority lists that are supplementary to the current L lists.

5.2 Solution robust predictive schedule

The results in the previous section were primarily meant to show the quality of different reactive procedures, but as we stated before, their lack of proactivity might lead to less robust schedules. Starting from a proactive predictive schedule is highly recommended. In this section we will apply the reactive procedures on a solution robust predictive schedule that is obtained by applying the STC heuristic (Van de Vonder et al. 2005a) on the RCPSP solution obtained by the combined crossover algorithm of Debels & Vanhoucke (2006). Table 7 summarizes results for the priority list policies. Rescheduling without looking at robustness is illogical because the included proactivity would directly disappear. Consequently, we do not test non-robust SGSs.

Table 7: Average expected stability cost for proactive schedules

λ	RP	RS
EBST1	155.35	153.05
EBST2	162.94	161.05
LAN	179.14	743.21
LST1	202.39	1031.43
LW	187.81	1182.61
RND	189.19	1178.26
EPST	159.58	164.58
MC	160.13	155.27

$EBST1$ is again the best priority list, but this time the robust serial SGS slightly outperforms the robust parallel SGS. However, serial SGSs remain far more sensitive to the used priority lists. The computational time required to apply the robust parallel SGS on a static priority list is still 0.04s per network instance, while the robust serial SGS is considerable faster (0.24s) on an STC schedule than on a quality robust schedule (0.52s).

Table 8: Advanced heuristics on proactive schedules

	$\sum_{j \in N} w_j E\lvert s_j^0 - s_j^T \rvert$	Time (s)
Sampling	148.56	12.2
TW Sampling	148.37	10.8
ILS	148.89	15.1

Also the sampling approach is adapted for use on proactive predictive schedules. Again, only robust SGSs will be considered. This results in 22 instead of 44 candidate projected schedules at every decision time. Considering Table 8, the sampling procedure now yields an average stability cost of 148.56 in on average 12.2s. This is a 4.37% improvement compared to our benchmark (λ_{EBST1},

RP) policy. *TWSampling* with $\Theta = 5$ slightly improves this result to 148.37. Surprisingly at first sight, *TWSampling* runs in less computational time (10.8s) than basic sampling on proactive schedules. The robust serial SGS still needs the generation of two complete schedules at any decision time, but as we stated above, it runs substantially faster on proactive schedules, making the robust parallel SGS the bottleneck. This SGS requires more computational time when a bad priority list is used. Because *TWSampling* always switches to $EBST1$ at $t + \Theta$, it becomes computationally more efficient than basic sampling.

The Iterated Local Search procedure is also considerably faster on a proactive schedule because it stops whenever there have been *max_without_imp* iterations without improvement and improvement will now be much harder to achieve. It runs on 15.1s and obtains an average stability cost of 148.89. In contrast to what we saw in Section 5.1, this is not better than *Sampling* or *TWSampling*.

6 Conclusions and further research

Reactive project scheduling in a stochastic environment is a multi-stage decision process. In this paper we examined several reactive procedures to repair a project schedule whenever activities are disrupted during execution. The objective is to minimize the deviation between the predictive schedule S^0 and the realized schedule S^T.

Two new robust scheduling generation schemes are proposed, i.e. the robust parallel and the robust serial SGS. In a stochastic project environment, these robust SGS generate feasible projected schedules with low stability cost from a priority list. We have given an in-dept analysis of the advantages and drawbacks of robust and traditional SGSs.

An extensive computational experiment revealed that a priority list that orders activities in non-decreasing order of activity starting times in the predictive schedule scores best among several examined lists. Both robust parallel and robust serial SGSs result in qualitative projected schedules when applied on such priority lists.

However, applying a single priority list will at some decision times result in bad decision taking. There is no reason to assume that the same priority list should be used at every decision point. Dynamic priority lists constantly update lists by including new information, but they obtain disappointing results.

A better approach to make appropriate decisions at every decision time is to select the best out of several candidate projected schedules. *Sampling* and *TWSampling* use several priority lists and SGSs to generate these candidates. In doing so, they obtain a significant improvement compared to single priority lists on both quality robust and solution robust predictive schedules. *Sampling* performs best when policies are included that generate schedules that differ substantially from the overall good policies.

The Iterated Local Search somewhat uses the inverse logic than *Sampling*. Instead of considering a set of completely different solutions, ILS tries to escape from an initial solution by applying local searches. The performance of ILS is

highly dependent on the parameter settings. *Sampling* will in general result faster in better solutions, while ILS will have difficulties to escape from the initial solutions, but will ultimately obtain better results. When starting from a proactive baseline schedule, *Sampling* and ILS obtain very similar results.

In our framework, we assume that protecting a schedule against future disruptions is the sole responsibility of the proactive scheduling routine. In the planning phase, we construct a proactive predictive schedule that will anticipate most future disruptions. When this plan becomes infeasible during project execution, we try to repair this predictive schedule in the best possible way. None of these procedures has a proactive nature. Opportunities to include extra safety are not exploited. Developing reactive procedures that anticipate future disruptions is an interesting future research direction.

A second future research topic might be to apply a sampling approach on stochastic scheduling, i.e. to minimize the expected makespan. This would require to select at every decision time the candidate projected schedule with minimum makespan, rather than the one with minimum deviation from the predictive schedule. Other priority lists should be incorporated to do so.

Acknowledgements

This research has been partially supported by Project OT/03/14 of the Research Fund K.U.Leuven and project G.0109.04 of the Research Foundation - Flanders (F.W.O.-Vlaanderen)

References

Alvarez-Valdez, R. & Tamarit, J. (1989). *Heuristic algorithms for resource-constrained project scheduling: A review and an empirical analysis*. Elsevier, Amsterdam. pp 113–134. In R. Slowinski and J. Weglarz (Eds.), Advances in Project Scheduling.

Artigues, C., Michelon, P. & Reusser, S. (2003). Insertion techniques for static and dynamic resource-constrained project scheduling. *European Journal of Operational Research*, 149(2), pp 249–267.

Ballestín, F. (2006). When it is worthwhile to work with the stochastic RCPSP? *Journal of Scheduling, to appear*.

Ballestín, F. & Trautmann, N. (2006). A metaheuristic approach for the resource-constrained weighted earliness-tardiness project scheduling problem. *Working paper*.

Brucker, P. & Knust, S. (2006). *Complex Scheduling*. Springer.

Davenport, A. & Beck, J. (2002). A survey of techniques for scheduling with uncertainty. *Unpublished manuscript*.

Debels, D. & Vanhoucke, M. (2006). Future research avenues for resource constrained project scheduling: Search space restriction or neighbourhood search extension. *Research report*. Ghent University, Belgium.

Fernandez, A. A. & Armacost, R. L. (1995). The role of the non-anticipativity constraint in commercial software for stochastic project scheduling. *Computers and Industrial Engineering*, 31, pp 233–236.

Graham, R. L. (1966). Bounds on multiproccessing timing anomalies. *Bell System Technical Journal*, 45, pp 1563–1581.

Hagstrom, J. (1988). Computational complexity of PERT problems. *Computers and Operations Research*, 18, pp 139–147.

Herroelen, W., De Reyck, B. & Demeulemeester, E. (2000). On the paper "Resource-constrained project scheduling: notation, classification, models and methods" by Brucker et al.. *European Journal of Operational Research*, 128(3), pp 221–230.

Herroelen, W. & Leus, R. (2004). Robust and reactive project scheduling: a review and classification of procedures. *International Journal of Production Research*, 42(8), pp 1599–1620.

Herroelen, W. & Leus, R. (2005). Project scheduling under uncertainty – Survey and research potentials. *European Journal of Operational Research*, 165, pp 289–306.

Kéri, A. & Kis, T. (2005). Primal-dual combined with constraint propagation for solving rcpspwet. *Proceedings of 2nd Multidisciplinary International Conference on Scheduling: Theory and Applications, New York*.

Kolisch, R. (1996). Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation. *European Journal of Operational Research*, 90, pp 320–333.

Kolisch, R. & Sprecher, A. (1997). PSPLIB - a project scheduling library. *European Journal of Operational Research*, 96, pp 205–216.

Leus, R. (2003). *The generation of stable project plans*. PhD thesis. Department of applied economics, Katholieke Universiteit Leuven, Belgium.

Leus, R. & Herroelen, W. (2004). Stability and resource allocation in project planning. *IIE transactions*, 36(7), pp 1–16.

Leus, R. & Herroelen, W. (2005). The complexity of machine scheduling for stability with a single disrupted job. *Operations Research Letters*, 33, pp 151–156.

Lourenço, H. R., Martin, O. & Stützle, T. (2002). *Iterated Local Search*. Kluwer, Boston. pp 321–353. In F. Glover and G. Kochenberger (eds.): Handbook of metaheuristics.

Möhring, R., Radermacher, F. & Weiss, G. (1984). Stochastic scheduling problems I - Set strategies. *Zeitschrift für Operations Research*, 28, pp 193–260.

Möhring, R., Radermacher, F. & Weiss, G. (1985). Stochastic scheduling problems II - General strategies. *Zeitschrift für Operations Research*, 29, pp 65–104.

Policella, N., Oddi, A., Smith, S. & Cesta, A. (2004). Generating robust partial order schedules. *In Proceedings of CP2004, Toronto, Canada*.

Schwindt, C. (2000). *Minimizing earliness-tardiness costs of resource-constrained projects*. Springer, Berlin. In Inderfurth K, Schwödiauer G, Domschke W, Juhnke F, Kleinschmidt P, Wäscher G (eds) Operations Research Proceedings.

Stork, F. (2001). *Stochastic Resource-Constrained Project Scheduling*. PhD thesis. Technical University of Berlin, School of Mathematics and Natural Sciences.

Van de Vonder, S., Demeulemeester, E. & Herroelen, W. (2005a). Heuristic procedures for generating stable project baseline schedules. *In proceedings of Third Euro Conference for Young OR researchers and practitioners (ORP3), Valencia, Spain*. pp 11–19.

Van de Vonder, S., Demeulemeester, E. & Herroelen, W. (2005b). An investigation of efficient and effective predictive-reactive project scheduling procedures. *Journal of Scheduling, to appear*.

Van de Vonder, S., Demeulemeester, E., Herroelen, W. & Leus, R. (2006). The trade-off between stability and makespan in resource-constrained project scheduling. *International Journal of Production Research*, 44(2), pp 215–236.

Vanhoucke, M., Demeulemeester, E. & Herroelen, W. (2001). An exact procedure for the resource-constrained weighted earliness-tardiness project scheduling problem. *Annals of Operations Research*, 102, pp 179–196.

Yu, G. & Qi, X. (2004). *Disruption Management - Framework, Models and Applications*. World Scientific, New Jersey.